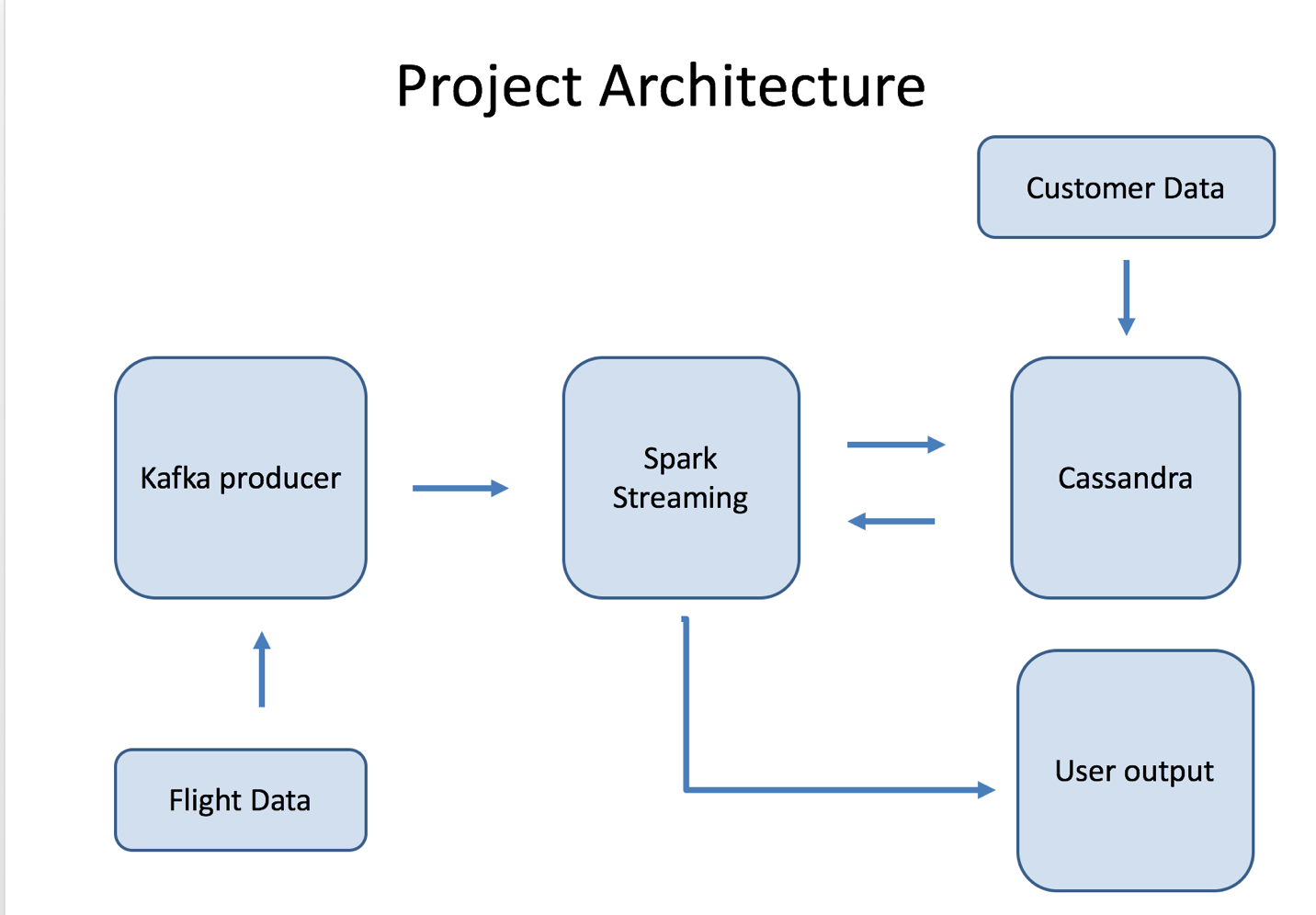
**Project overview/summary:**

**Technology summary:**

|  |  |  |
| --- | --- | --- |
| **Streaming** | **Kafka** | **AWS Cluster (1 zookeeper , 1 Broker)** |
| **Producer** | **Python Kafka** | **Deployed to Broker** |
| **Big data platform** | **Spark 2.1** | **AWS EMR (1 Master , 2 Core servers)** |
| **Consumer** | **Pyspark kafka** | **AWS EMR step executed client mode** |
| **Data storage** | **Cassandra 3.9, cqlsh 5.0.1** | **AWS Cassandra server** |
| **Data Analysis** |  |  |
| **Local desktop** | **Windows 10** |  |
| **Github** | **Code and file repository** | **https://github.com/mohanrkrishna/csci63-SparkstreamingIOT.git** |

****

**Infrastructure set up overview:**

* Cloud – AWS free account
* Kafka cluster

AWS Cloud formation used to create Kafka cluster with 1 zookeeper server and 1 Broker server; Cloud formation has a template to provision these servers pre-configured with all settings required for kafka streaming like adding zookeeper server in server.properties of kafka broker

SSH into zookeeper server to create 1 topic “airline1” for this project

More details on set up below

* Spark cluster

* Cassandra cluster

Bitnami image from marketplace which provides an out-of-the-box Cassandra cluster

**Design-Code overview:**

**Execution demo:**

**Infrastructure set up details(Kafka):**

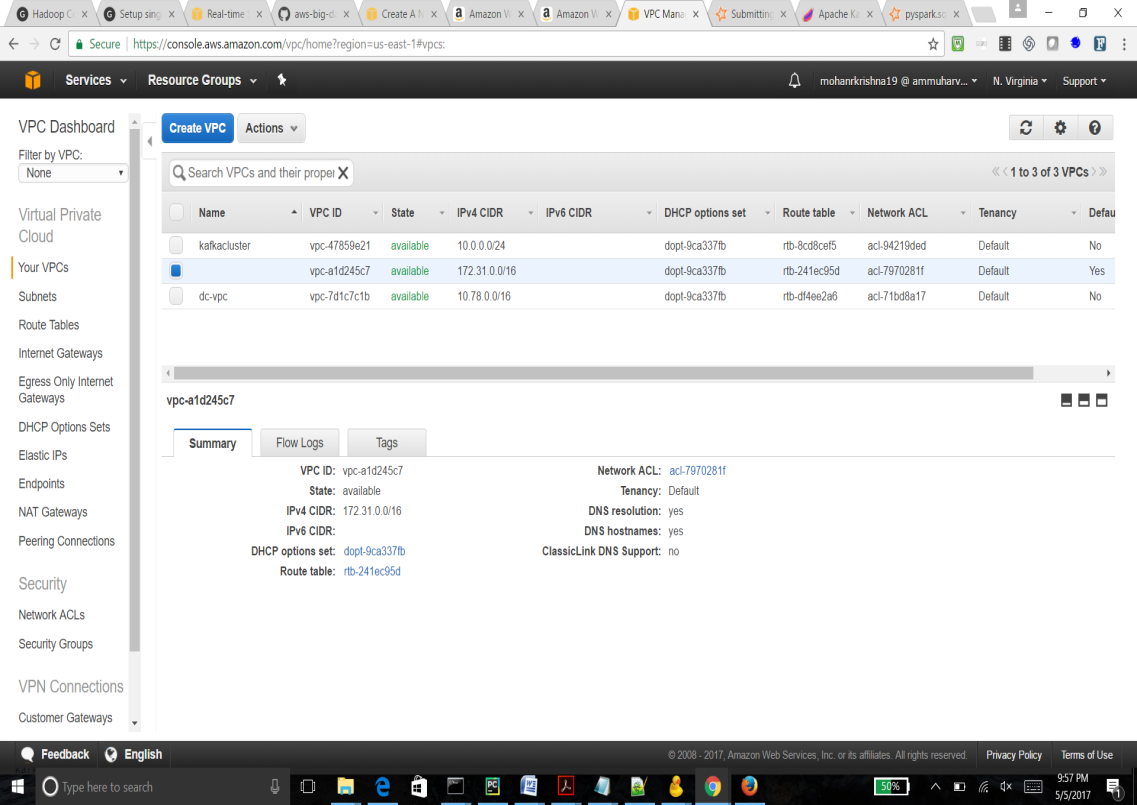
**Kafka cluster:**

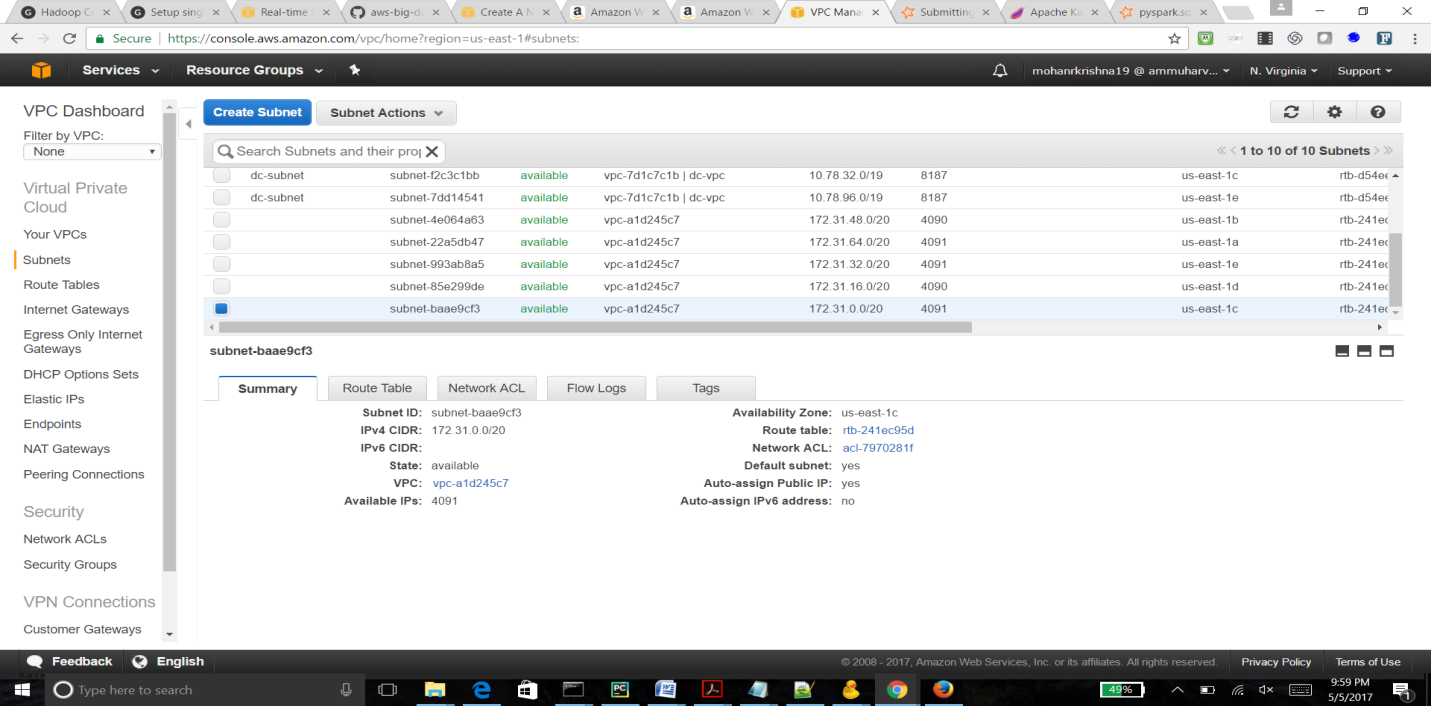
* Create following from AWS VPC to create Virtual private cloud and subnet for creating Kafka cluster
  + Create EC2 keyvalue pair for ssh; Download .pem file to local desktop
  + Create VPC and Subnet ; Important settings –

(VPC) DNS resolution: yes ; DNS hostnames:yes

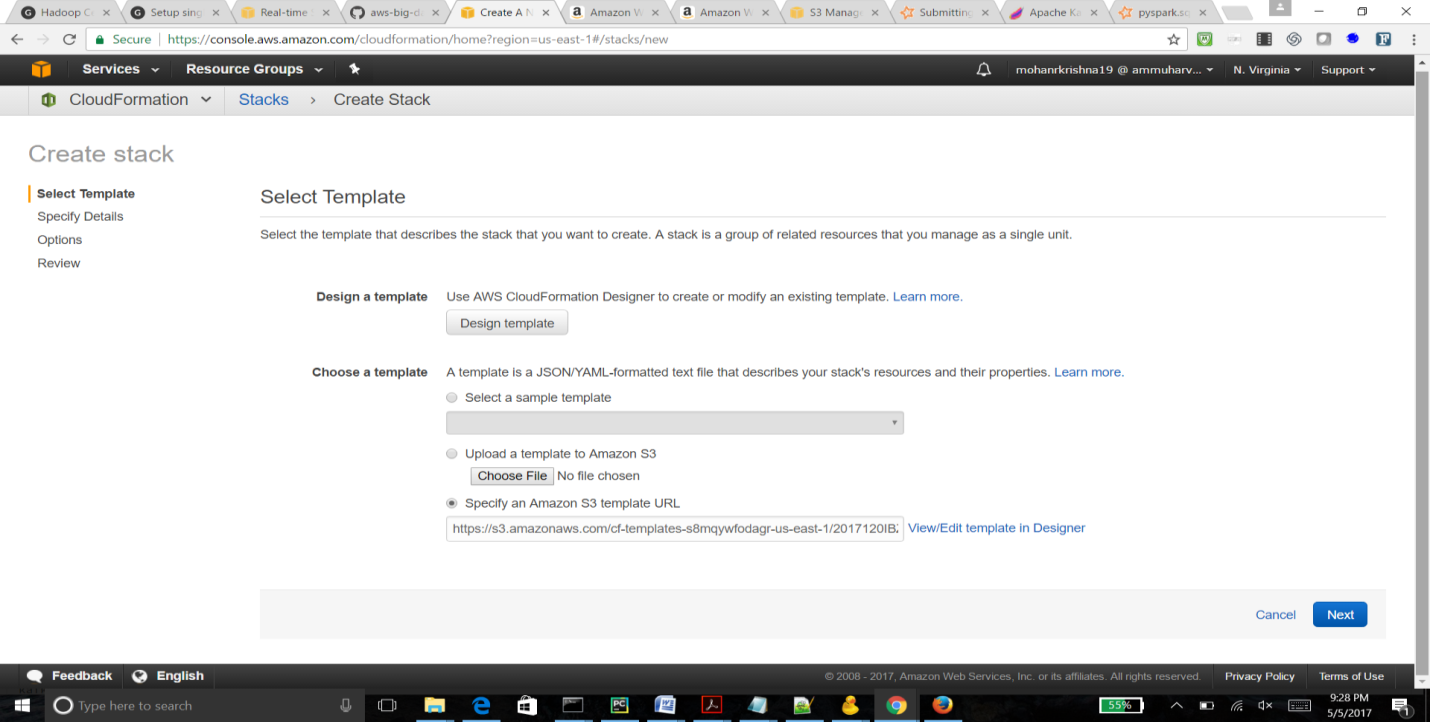
(Subnet) Auto-assign Public IP: Yes

|  |  |
| --- | --- |
|  |  |

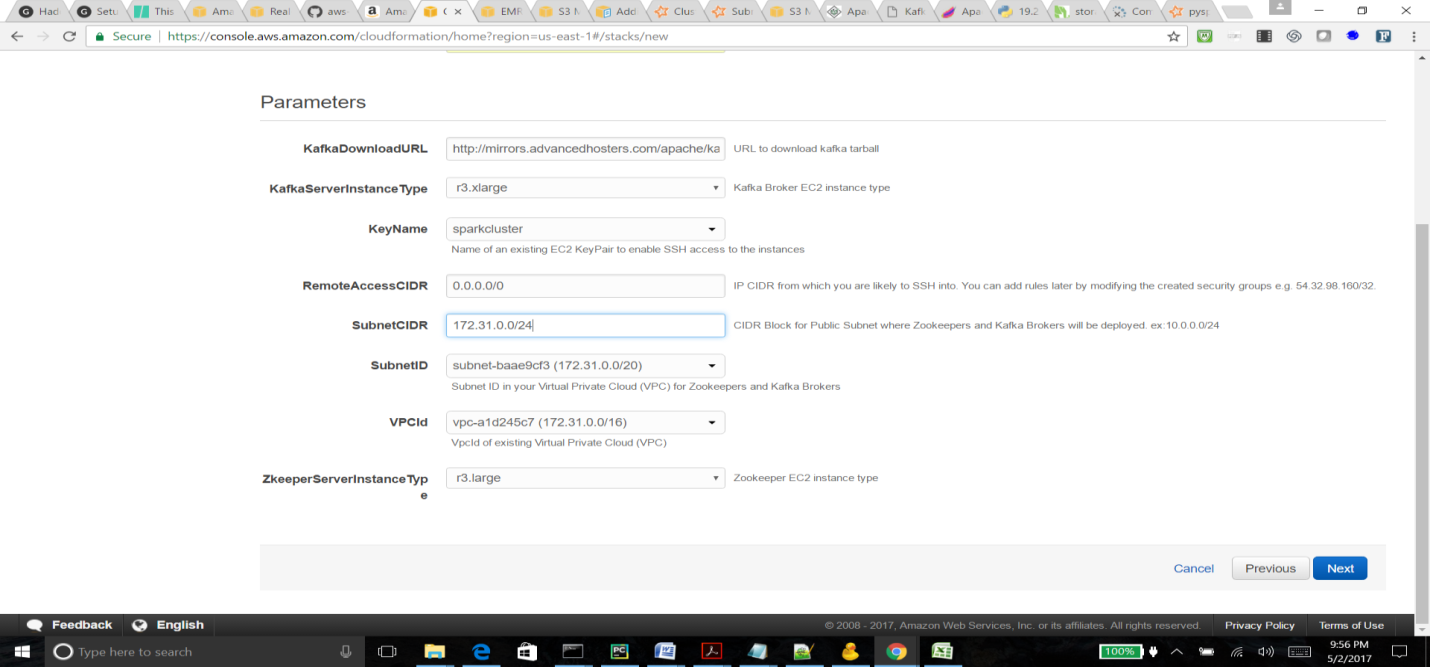




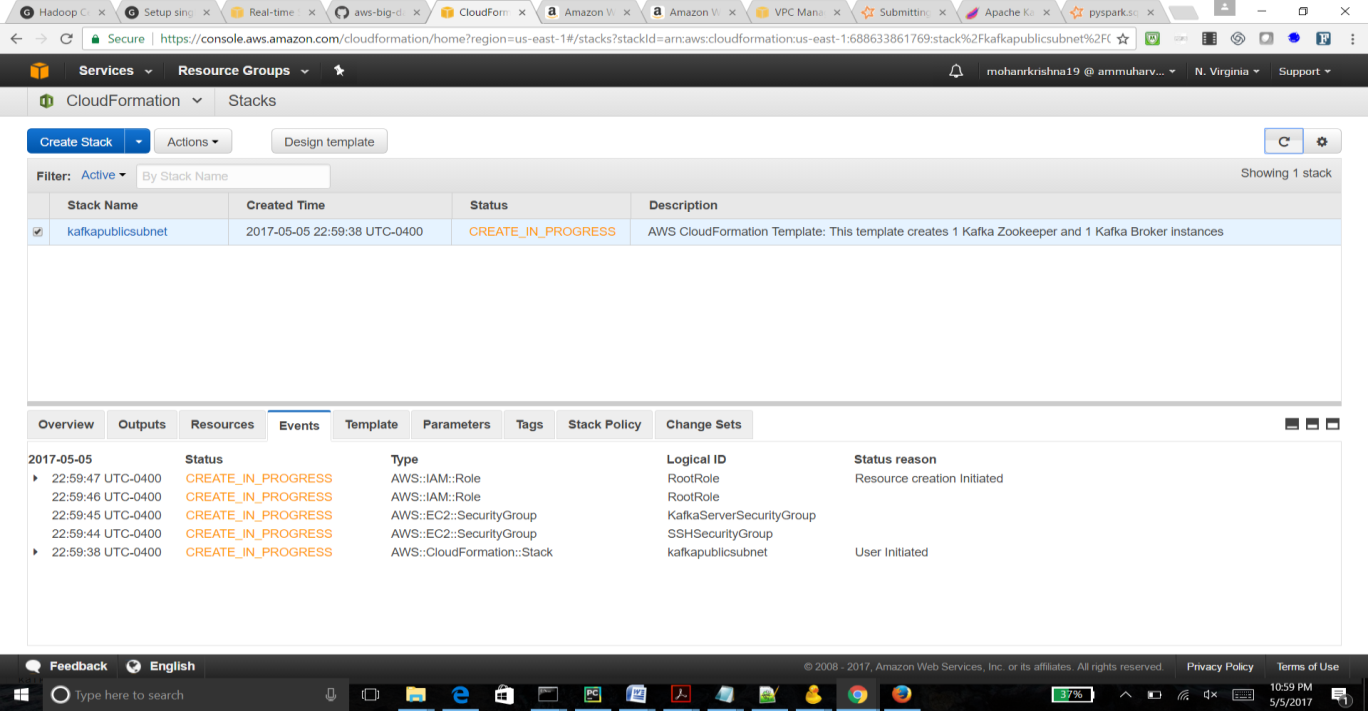
* Use json template at [CloudFormation template for public subnets](https://github.com/awslabs/aws-big-data-blog/blob/master/aws-blog-sparkstreaming-from-kafka/cloudformation/SetupKafka-AWS-PublicSubnets.json) ; Download to s3
* From AWS Cloud formation , create stack , choose above s3 json



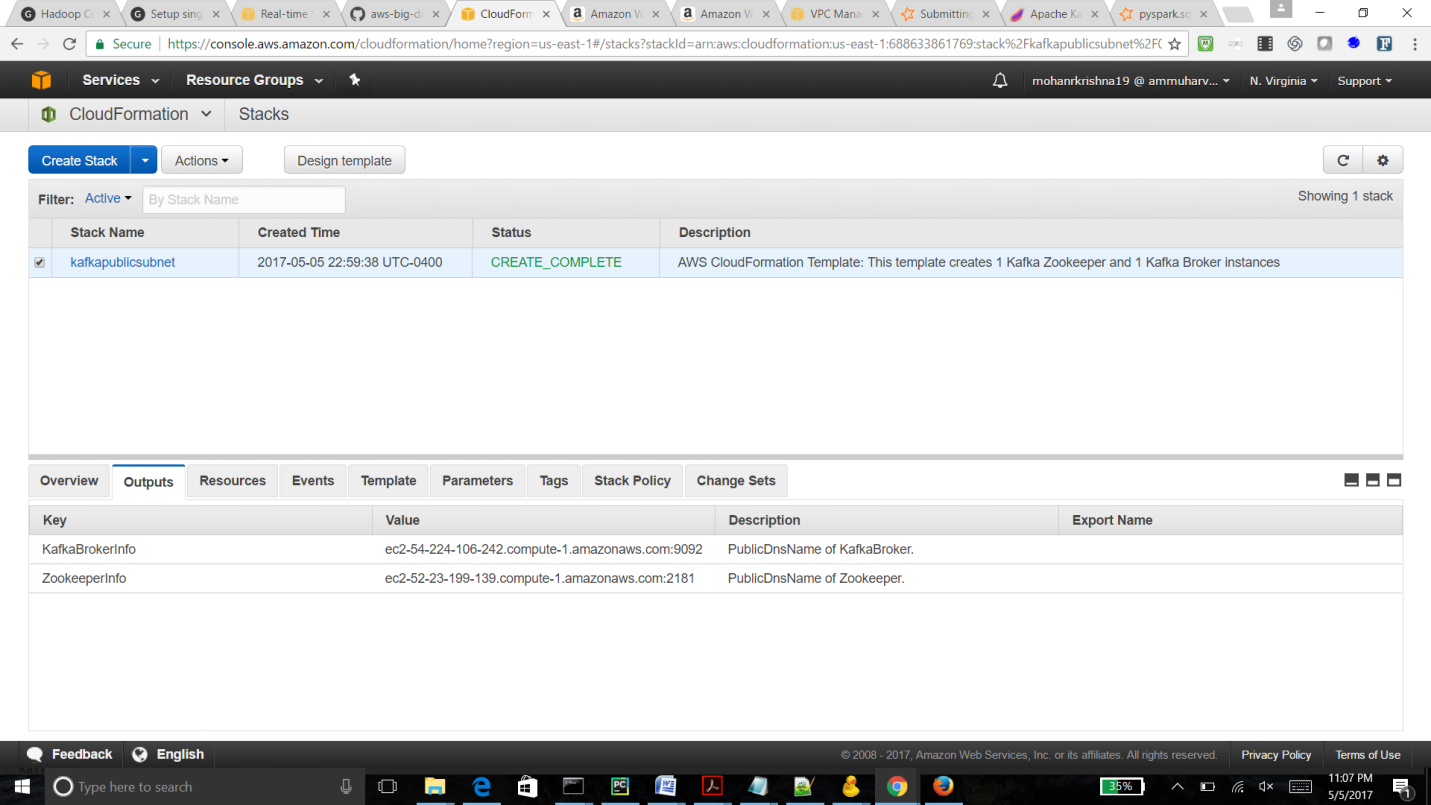
* Give “stackname” kafkapublicsubnet ; Select VPCid and subnet created above; Choose KeyName=EC2 keyvalue pair created. Important : SubnetCIDR should match Subnet CIDR , otherwise servers cannot be connected ; Keep Remote CIDR to 0.0.0.0/0 ; this is for laptops from which SSH is invoked. SourceIP can be added later too from EC2 menu



* Below screen shows progress , wait until ‘CREATE\_COMPLETE’



* Navigate to Outputs and save publicdns of KafkaBroker , Zookeeper servers , Required to login using SSH



**Kafka cluster create topic:**

* Install Git bash desktop app on windows
* Using Git bash , cd to directory with EC2 private key and remote login using SSH

email@LAPTOP-9CHTJ9IO MINGW64 /D/BigdataVMWare/Aws Keys

**$ ssh -i "sparkcluster.pem" ec2-user@ec2-52-23-199-139.compute-1.amazonaws.com**

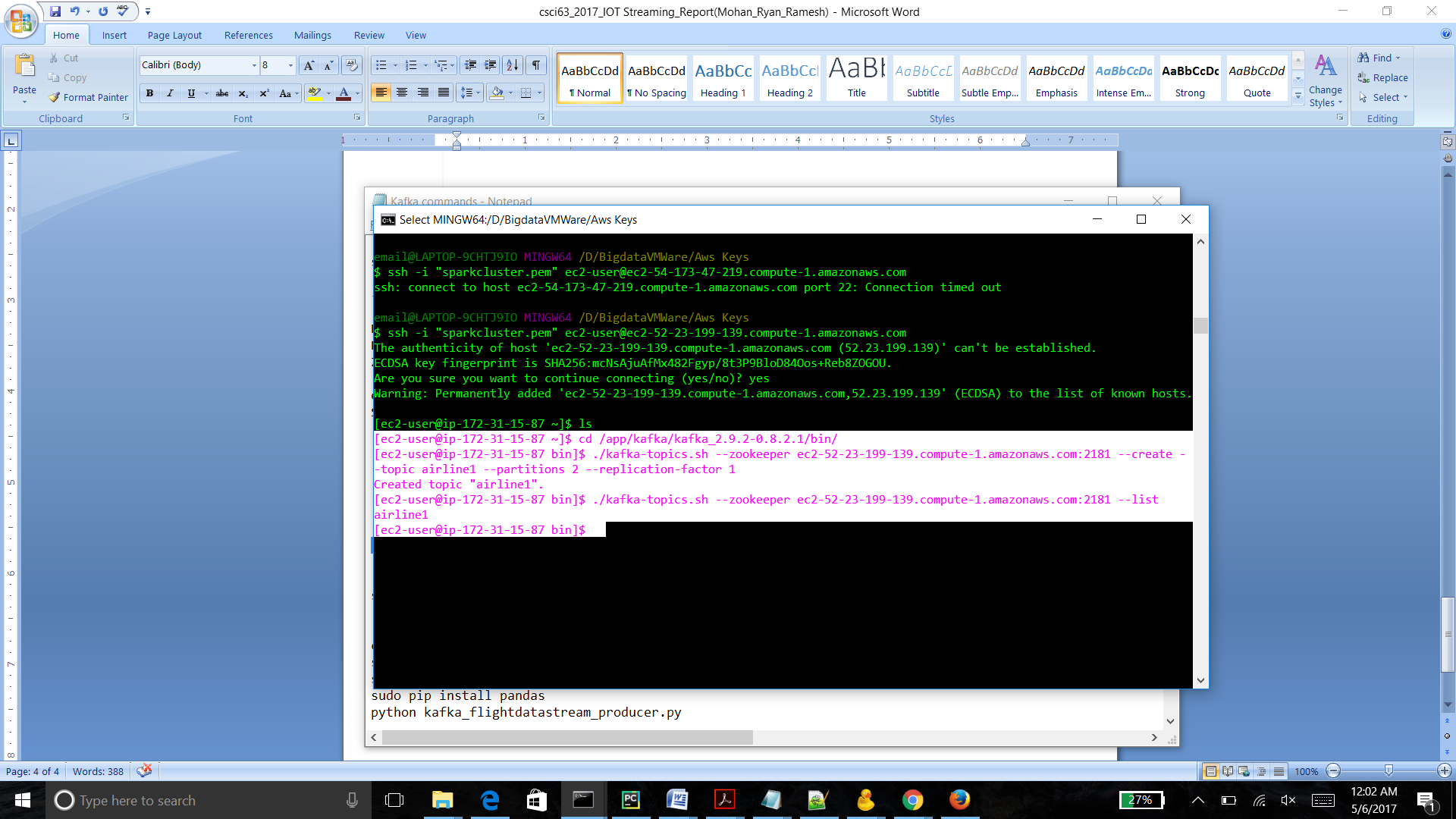
The authenticity of host 'ec2-52-23-199-139.compute-1.amazonaws.com (52.23.199.139)' can't be established.

ECDSA key fingerprint is SHA256:mcNsAjuAfMx482Fgyp/8t3P9BloD84Oos+Reb8ZOGOU.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'ec2-52-23-199-139.compute-1.amazonaws.com,52.23.199.139' (ECDSA) to the list of known hosts

* Navigate to kafka bin directory and create topic ; Verify topic created using --list

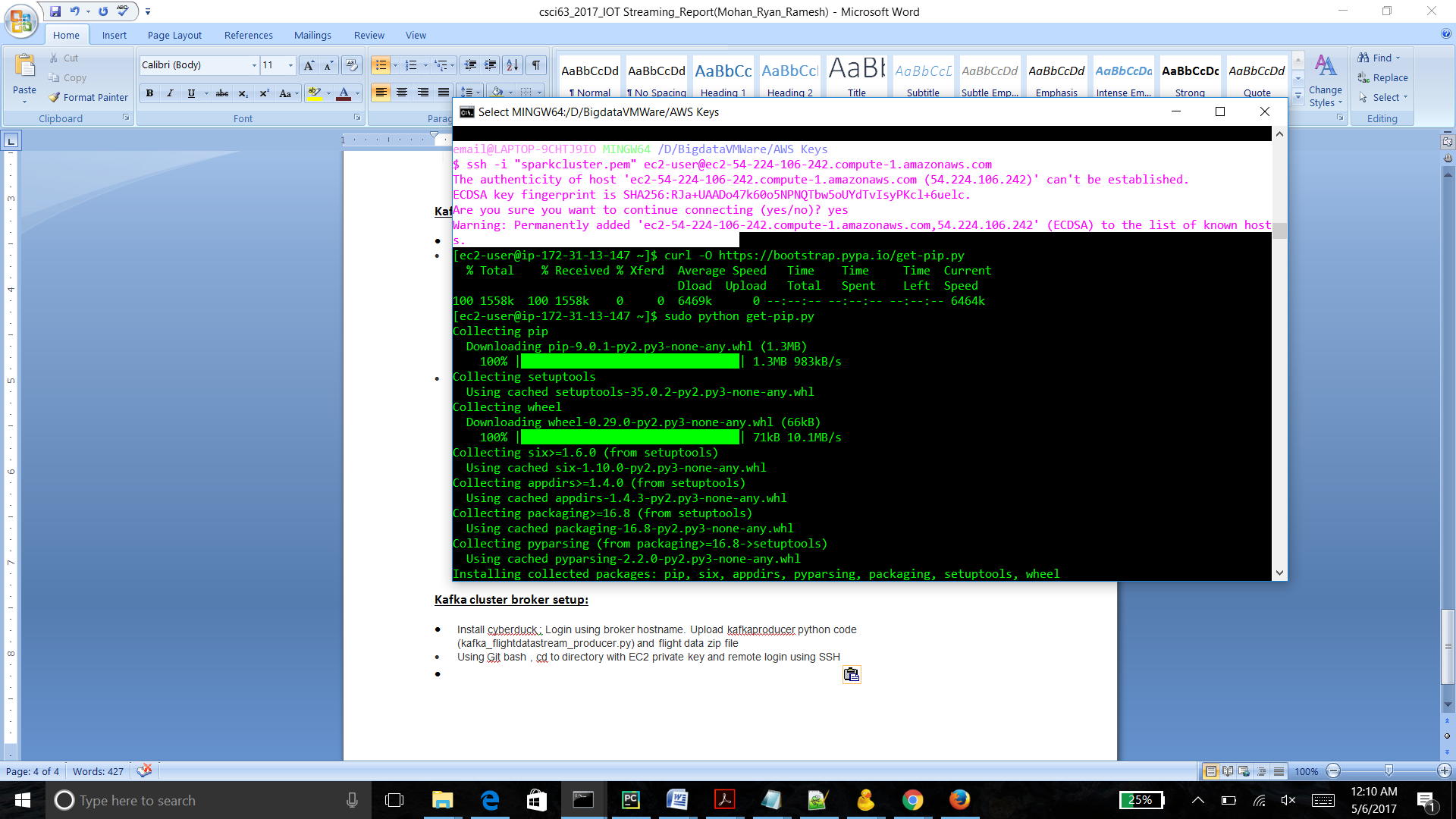


**Kafka cluster broker setup:**

* Install cyberduck ; Login using broker hostname. Upload kafkaproducer python code (kafka\_flightdatastream\_producer.py) and flight data zip file
* Using Git bash , cd to directory with EC2 private key and remote login using SSH

**email@LAPTOP-9CHTJ9IO MINGW64 /D/BigdataVMWare/AWS Keys**

**$ ssh -i "sparkcluster.pem"** [**ec2-user@ec2-54-224-106-242.compute-1.amazonaws.com**](mailto:ec2-user@ec2-54-224-106-242.compute-1.amazonaws.com)



* Kafka broker is used to run producer python program. So below python packages are required. Pip needs to be installed first followed by kafka-python , pandas

curl -O https://bootstrap.pypa.io/get-pip.py

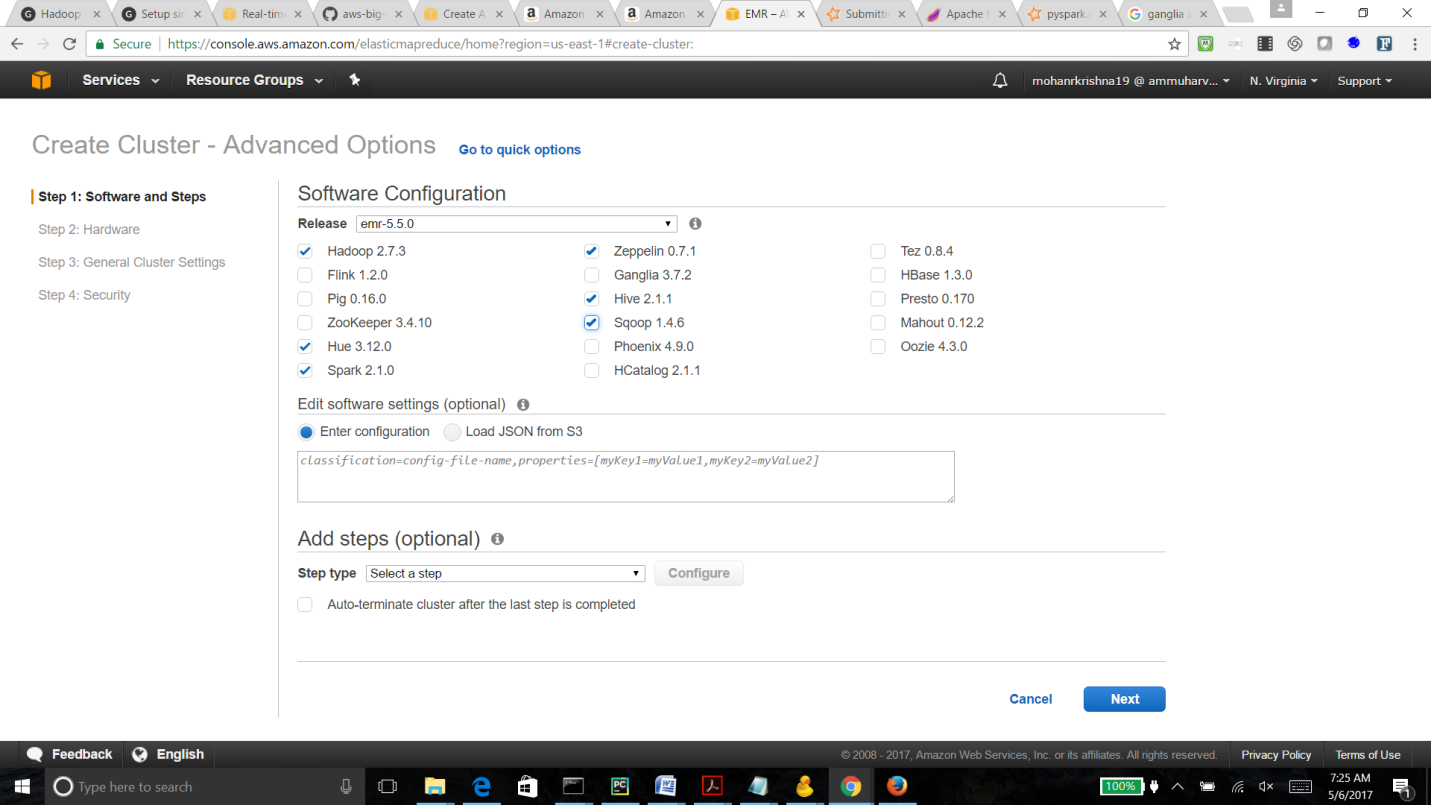
sudo python get-pip.py

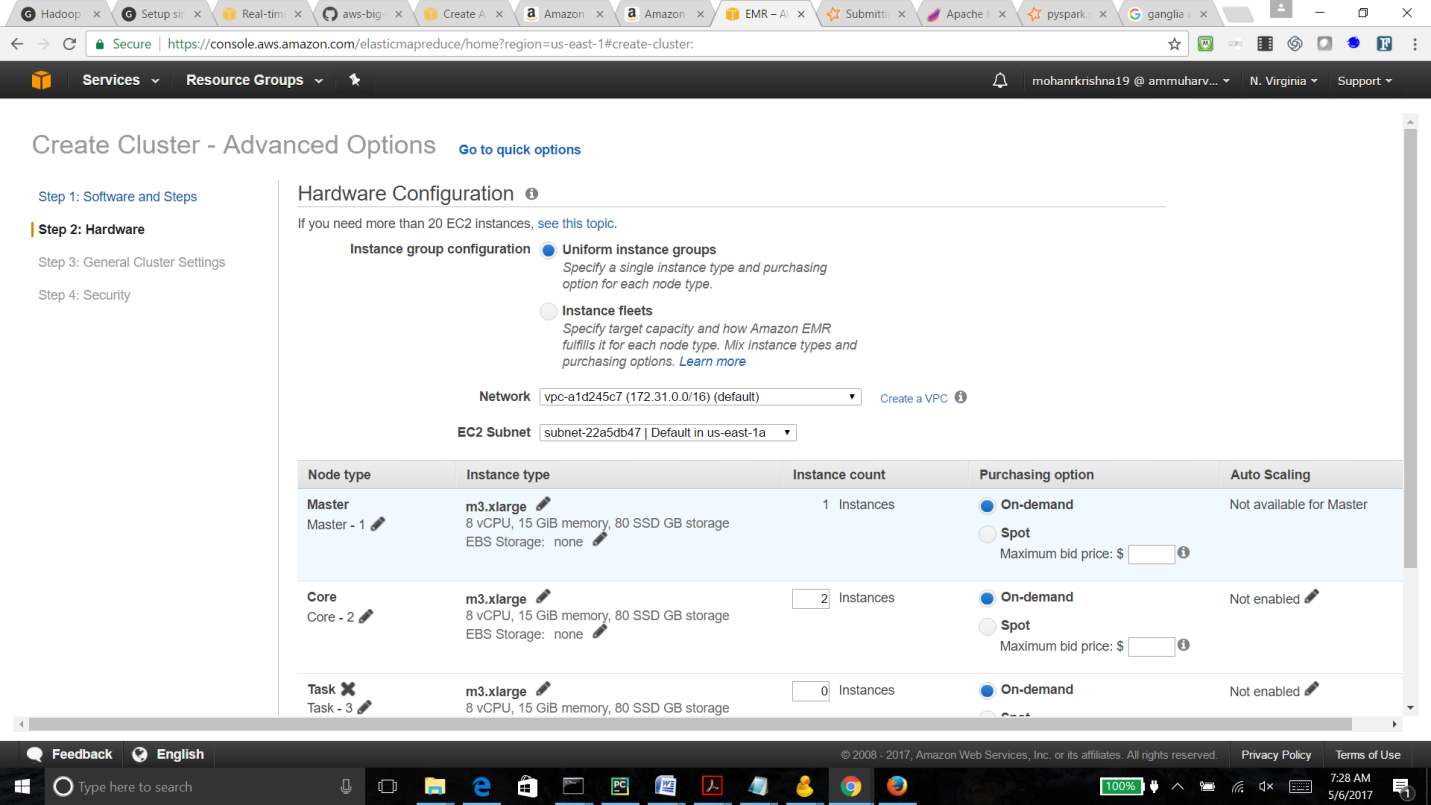
sudo pip install kafka-python

sudo pip install pandas

**Infrastructure set up details(Spark):**

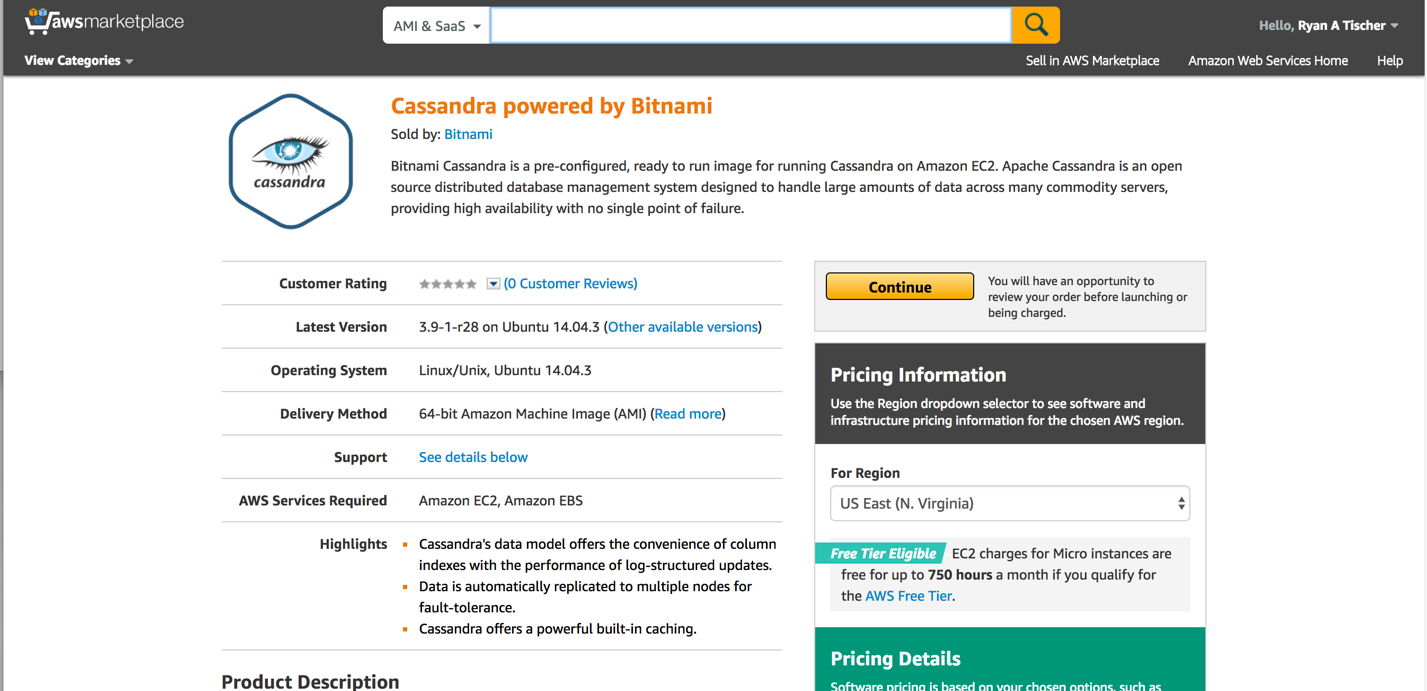
* Navigate to Amazon EMR (AWS)
* Click Create cluster ; give name of cluster – ‘sparkcluster’ and go to advanced options
* Choose latest emr – emr-5.5.0 , hadoop is default selected and required for spark to use HDFS , Yarn ; Choose Spark 2.1.0



* Next hardware configuration , as this is small scale and pilot choose low scale servers to avoid charges ; Master node – m3.xlarge(8core , 15GB) , 2 Cores (for Spark Worker nodes, 2 Core , 3.8GB) ; Choose Network VPC and Subnet same as Kafka cluster to be able to avoid cross network access issues between spark and kafka servers
* Next general options , leave as-is
* Next Security Options , choose EC2 key value pair. I chose the same one used for Kafkacluster; Suggested to use different one
* Click Create cluster; It would take 15-20 mins

**Infrastructure set up details(Cassandra):**

Cassandra server is a pre-provisioned instance by Bitnami. The Bitnami image features Cassandra 3.9 running on Amazon machine image (Amazon Linux)



Connecting to the image requires a certificate file, in this case Tischer.pem. The certificate authenticates the session with the following command:

ssh -i "tischer.pem" [ubuntu@ec2-54-226-206-176.compute-1.amazonaws.com](mailto:ubuntu@ec2-54-226-206-176.compute-1.amazonaws.com)

Once connected access to Cassandra via the following command:

cqlsh -u cassandra -p KSns9DfaYahq

In this case the username is ‘cassandra’ and ‘KSns9DfaYahq’ is the password

**Cassandra configuration**

Cassandra is configured for a single keyspace called flightkeyspace

To following command creates the keyspace

CREATE KEYSPACE flightkeyspace WITH REPLICATION = { 'class':'SimpleStrategy', 'replication\_factor':1};

The following command uses the keyspace

USE flightkeyspace;

The following tables are created within the flightkeyspace keyspace.

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| flightiot | Holds flight IOT data, populated from Spark/kafka |
| customers | Holds customers on flights. Populated with Mock data |
| City\_data | Holds data about citys |

Tables are manually created using the following command. (Example for one table only.

create table city\_data (OriginAirportID int,

DestAirportID int,

destcityname text,

OriginCityName text,

origin text,

dest text,

PRIMARY KEY ((OriginAirportID, DestAirportID, destcityname, OriginCityName, origin, dest)));

In the event of moving data to or from CSV the following command was used:

copy flight\_test (OriginAirportID, DestAirportID, destcityname, OriginCityName origin, dest) to ‘temp12.csv’

Flight IOT table is populated from the streaming data. It is not meant for human consumstion. FlightIOT has the following columns.

uniquecarrier | flightnum | flightdate | origin | dest | actualelapsedtime | airlineid | airtime | arrdel15 | arrdelay | arrdelayminutes | arrivaldelaygroups | arrtime | arrtimeblk | cancellationcode | cancelled | carrier | carrierdelay | crsarrtime | crsdeptime | crselapsedtime | dayofmonth | dayofweek | departuredelaygroups | depdel15 | depdelay | depdelayminutes | deptime | deptimeblk | destairportid | destairportseqid | destcitymarketid | destcityname | deststate | deststatefips | deststatename | destwac | distance | distancegroup | diverted | firstdeptime | flights | lateaircraftdelay | longestaddgtime | month | nasdelay | originairportid | originairportseqid | origincitymarketid | origincityname | originstate | originstatefips | originstatename | originwac | quarter | securitydelay | tailnum | taxiin | taxiout | totaladdgtime | weatherdelay | wheelsoff | wheelson | year

**and example data**

AS | 657 | 2017-01-03 00:00:00.000000+0000 | SAT | SEA | 271 | 19930 | 251 | 0 | -16 | 0 | -2 | 2102 | 2100-2159 | 0 | 0 | AS | 0 | 2118 | 1840 | 278 | 3 | 2 | -1 | 0 | -9 | 0 | 1831 | 1800-1859 | 14747 | 1474703 | 30559 | Seattle, WA | WA | 53 | Washington | 93 | 1774 | 8 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 14683 | 1468303 | 33214 | San Antonio, TX | TX | 48 | Texas | 74 | 1 | 0 | N472AS | 6 | 14 | 0 | 0 | 1845 | 2056 | 2017

B6 | 1884 | 2017-01-26 00:00:00.000000+0000 | MCO | JFK | 140 | 20409 | 116 | 0 | -12 | 0 | -1 | 1517 | 1500-1559 | 0 | 0 | B6 | 0 | 1529 | 1301 | 148 | 26 | 4 | -1 | 0 | -4 | 0 | 1257 | 1300-1359 | 12478 | 1247803 | 31703 | New York, NY | NY | 36 | New York | 22 | 944 | 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 13204 | 1320402 | 31454 | Orlando, FL | FL | 12 | Florida | 33 | 1 | 0 | N907JB | 4 | 20 | 0 |

Customers table holds customer data is populated with Mock data from <https://www.mockaroo.com/>. Customers table has the following columns

flightnum | email | originairportid | destairportid | first\_name | id | last\_name | phone

and example data

flightnum | email | originairportid | destairportid | first\_name | id | last\_name | phone

-----------+-----------------------------+-----------------+---------------+------------+-----+-------------+-------------------

44 | ntibb22@prweb.com | 11042 | 11298 | Nissy | 75 | Tibb | 502-(644)575-3323

3042 | tdibbs71@123-reg.co.uk | 11603 | 12892 | Terese | 254 | Dibbs | 63-(350)531-3415

44 | rshepherdson15@cdc.gov | 11042 | 11298 | Rickie | 42 | Shepherdson | 7-(658)575-6810

3042 | owestmacott6p@dell.com | 11603 | 12892 | Orella | 242 | Westmacott | 81-(313)688-3710

4822 | jybarradp@toplist.cz | 12278 | 13487 | Janessa | 494 | Ybarra | 33-(422)984-2240

4674 | fhaughtonbe@soup.io | 13264 | 14869 | Fredrika | 411 | Haughton | 46-(477)414-8251

494 | cheyburn3@howstuffworks.com | 11697 | 11057 | Cally | 4 | Heyburn | 7-(434)411-8974

4822 | aforsythdc@webnode.com | 12278 | 13487 | Antonino | 481 | Forsyth | 86-(479)926-4568

44 | cwavell1o@dmoz.org | 11042 | 11298 | Cristionna | 61 | Wavell | 63-(765)701-1887

511 | awatting44@sfgate.com | 11298 | 11042 | Antin | 149 | Watting | 86-(764)715-5868

City Data holds City names and airport codes. It is populated from BTS data. City\_data table has the following columns

originairportid | destairportid | destcityname | origincityname | origin | dest

and example data

originairportid | destairportid | destcityname | origincityname | origin | dest

-----------------+---------------+------------------+--------------------+--------+------

14683 | 13487 | Minneapolis, MN | San Antonio, TX | SAT | MSP

10397 | 14633 | Rochester, MN | Atlanta, GA | ATL | RST

11618 | 13487 | Minneapolis, MN | Newark, NJ | EWR | MSP

12173 | 14831 | San Jose, CA | Honolulu, HI | HNL | SJC

13198 | 12339 | Indianapolis, IN | Las Vegas, NV | MCI | IND

14869 | 14679 | San Diego, CA | Salt Lake City, UT | SLC | SAN

12892 | 12339 | Indianapolis, IN | Los Angeles, CA | LAX | IND

14193 | 11042 | Nashville, TN | Pensacola, FL | PNS | CLE

10397 | 13485 | Madison, WI | Atlanta, GA | ATL | MSN

14122 | 12266 | Houston, TX | Pittsburgh, PA | PIT | IAH